Center for Applied Autonomous Sensor Systems (AASS)
Örebro University, School of Science and Technology
SE-70182 Örebro
Sweden

# OOP Lab III: Classes

In this lab we start converting our OO design into the code!

## Exersize I: Tune your settings

In this course, we look at the newer C++ standards (C++11 and up) and it is essential to make sure that the compiler used in NetBeans is at least C++ 11 (or better C++ 14, or even better C++17 if available). In order to enforce this, do the following:

**Right click on Project -- Properties -- Build (open it) -- C++ Compiler -- (on the right side) Basic Options -- C++ Standard -- C++ 11 or C++ 14.**

Note: *C++ 17 was released in December 2017 and compilers that conform to this standard are already available. Keep tuned with the latest language updates to switch quickly!*

## Exersize II: Re-invent the Wheel

Create a class **Wheel** with the following members:

- Data Members
    - double radius;
    - double rpm,
    - double angle;
    - double pressure;
- Member Function
    - Non-default constructor to create a Wheel with radius;
    - Getter and setter functions for rpm, return bool from setter;
    - Getter and setter functions for angle, return bool from setter;
    - Getter for radius;

Advise: be restrictive with your data members. Especially, do not make data members public, unless absolutely sure in that.

Advise: In NetBeans, if you make a right click on the source files, and then ask it to create a new C++ class, you will get perfectly formatted header and source files with the basic structure already in place.

Note: getters and setters are just normal functions, which allow you to get and set value of a private member. For example:

```
class Timer {
```

```
    int h;
public:
    int getH() {
        return this -> h;
    }
    void setH(int h) {
        this -> h = h;
    }
};
```

In the examle, getH is a getter and setH is a setter. Isn't is cool that you can expose your private data member using public member functions? ;)

Instantiate a wheel in your main program.

## Exersize III: Make Friends

In your Wheel class, make sure that pressure is private.

Create a new class Pump. In this class, declare and define a function setPressure, which takes as arguments a wheel and a double value for pressure.

In the main program, use Pump class to set pressure in the wheel.

## Exersize IV: Compare Wheels

In your wheel class, define at least two comparizon operators (for example, ==, !=, <, >, <=, >=). The operators should compare the current speed of two wheels by using radius and rpm. Operators are defined with the keyword “operator”. Please, refer to the course book for more details.

In the main program, instantiate several wheels with some values, and compare them.

## Exersize IV: Create a Car

Create a class Car with with the following data members and member functions:

- Data Members
    - Color color; //Color is enum
    - int weight;
    - string model;
    - double linearVelocity;
    - double angularVelocity;
    - Wheel * flWheel; //Assume that Wheel is an external class


    - Wheel * frWheel;
    - Wheel * rlWheel;
    - Wheel * rrWheel;
- Member Function
    - bool setSpeed(double speed);
    - bool setTurningAngle(double angle);

Note: *it is a good idea to return some value from the setter functions. For example, if the desired speed is impossible to set, the function might return false. Or, you can return some error code, or 0 (zero) if everything is fine.*

Question: for the Car class, you need to specify an enumeration for color. Think where would you place it in your project!

Example of Color:

```
enum Color { red, green, blue };
Color r = red;
switch(r)
{
  case red  : std::cout << "red\n";   break;
  case green: std::cout << "green\n"; break;
  case blue : std::cout << "blue\n";  break;
}
```

In the main program, create three cars. Create them in the way to invoke default constructor, copy constructor, and copy assignment operator.

Place a breakpoint in the beginning of your main program and debug step by step looking at variables, and stack. At the moment when a new car is created what values are in linear and angular velocities? Put the answer in a comment in your program.Make a commit.

## Exersize V: Custom Constructors

In the previous program, add a non-default constructor to the Car class, which would create a new car with a given color. Try to run your program. Does it compile? What should be done to keep the non-default constructor and make the program to work again? Make your car great again! :)

Now, as soon as you defined some constructor(s) in your Car class, define explicitly all other Special Member Functions. Each of them should output a phrase to the console, saying which constructor is called and the model of the car. Empty model name? Create a setter/getter member functions in you class and give each car a unique name! Test

constructors in the main function!

Make a commit.

## Exercise VI: Shallow vs Deep

When you defined a copy constructor for your Car class, how did you copy the wheels?

Make a member function in Car class, which would print out current angle and RPM for each wheel. Additionally, fill the setSpeed and setTurningAngle with code to set the same speed and the same angle the wheels (regardless the physical possibility of doing that).

Now, in the main program, create a car and make another one by copying the first (invoke copy constructor). Set different speeds and turning angles of both cars. Print out the current angle and rpm of all wheels of each car. Are they different or the same?

Further in your program, create your first car with a pointer, make the second as a copy, and delete the first. Now, try to get angles and rpm of the wheels of the second car. What happens? Why?

Implement a deep copy constructor for your car class.

(Optional) Ackermann kinematics is the modern steering mechanism in cars. Implement setSpeed and setTurningAngle functions to set correct angle and rpm for each wheel according to Ackermann steering. Read more about it here: https://en.wikipedia.org/wiki/Ackermann_steering_geometry